ELSEVIER

Computer Communications 24 (2001) 667–676

computer communications

www.elsevier.com/locate/comcom

Ordered lookup with bypass matching for scalable per-flow classification in layer 4 routers

Ying-Dar Lin*, Huan-Yun Wei, Kuo-Jui Wu

Department of Computer and Information Science, National Chiao-Tung University, 1001 Ta Hsueh Road, Hsinchu, Taiwan, ROC

Received 4 January 2000; accepted 27 July 2000

Abstract

In order to provide different service treatments to individual or aggregated flows, layer 4 routers in Integrated Services networks need to classify packets into different queues. The classification module of layer 4 routers must be fast enough to support gigabit links at a rate of millions of packets per second. In this work, we present a new software method OLBM to lookup multiple fields of a packet, in a dynamically pre-defined order, against the classification database. This algorithm also uses a technique called bypass matching and can classify packets at a rate of well over one million packets per second while scaling to support more than 300k flows. Complexity analysis and experiment measurements are also presented in this study.

Keywords: Classification; Layer 4 router; Packet filtering; Lookup; Match; Scalability

1. Introduction

In order to support QoS in Integrated Services (IntServ) [1–3] networks, several traffic control modules need to be added into the layer 4 routers that examine not only IP headers but also transport-layer headers. The admission control, in the control-plane, and the classifier and the scheduler, in the user-plane, are three basic modules for QoS traffic control. The classifier, which *distinguishes* an incoming packet into different flows, becomes essential. Besides QoS processing, firewall and VPN, [4] for example, also need the classifier to classify packets based on multiple fields. In this work, we focus on the classification for per-flow QoS processing.

There are three key components in the classification module: the filter database, the classification database, and the classifier. The filter database consists of filtering rules updated by the admission control module at run-time. Then the filter database inserts its information to the classification database as search indexes for the classifier to refer. Fig. 1 shows the role of the classification module and its process. Once a packet comes to a classifier, the classifier checks the five fields against the existing classification database. A packet is said to *match* a filtering rule if the values of all the five fields in the packet are exactly the same as those specified by the filtering rule. If a matched filtering rule is found, the packet is put into the corresponding queue for special processing.

Three methods have been proposed for fast classification. One is hardware based, which uses the hardware parallel processing power for multi-dimension range matching [5], and the other two are software based. The one of Ref. [6] combines destination–source tries and cross-producting, while that of Ref. [7] take TSS (Tuple Space Search) as its main technique. Table 1 is a summary comparing these three methods.

All these methods lookup *all* the five fields of a packet against *each* filtering rule. In addition, they do not seem to be scalable enough to meet the high scalability requirement. Thus, we provide a scalable method: *Ordered Lookup* with *Bypass Matching* (OLBM). Ordered Lookup (OL) may save unnecessary work without looking up all the five fields. Bypass Matching (BM) can help to finish the OL more quickly.

The rest of this work is organized as follows. We give our design objectives and motivation in Section 2. Section 3 presents the OLBM algorithm. Section 4 draws the analytical results of the worst case. Experimental performance studies, in terms of memory usage, throughput, sensitivity to locality, lookup order, scalability, and extensibility, are presented in Section 5. Finally, a conclusion and future work are given in Section 6.

* Corresponding author. Tel.: +886-3-573-1899; fax: +886-3-572-1490.
E-mail address: ydlin@cis.nctu.edu.tw (Y.-D. Lin).

0140-3664/01/$ - see front matter
PII: S0140-3664(00)00305-4

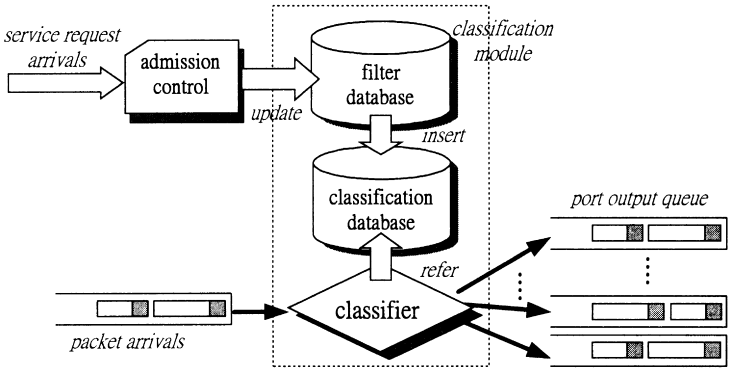


Fig. 1. The role of the classification module and its process.

2. Objectives and motivation

2.1. The design objectives of a classification algorithm

There are three objectives for designing a classification algorithm:

1. *Throughput.* The algorithm must be able to process at least one million packets per second. For an OC-3 link of 155 Mbps, considering that all incoming packets are as small as 64 bytes, the classifier must process 317,440 packets in 1 s. Thus, for a router with multiple interfaces, the processing rate of over one million packets per second is required.
2. *Scalability.* The algorithm must be scalable. Recent studies have shown that an OC-3 link might have an average of 240k flows [8], which implies that there would be as many as 240k filtering rules in the data structures of a classifier.
3. *Extensibility.* The algorithm must be flexible enough to be extended to lookup using more fields, or even payload, against IP prefix type filters.

2.2. The design motivation of ordered lookup

The motivation for us to design OL is that the classifier could use fewer fields of the packet header to find the matched filtering rule, if any. It needs fewer memory references and CPU instructions, which means that the classifier can achieve the same function faster.

This algorithm is designed for software implementation. Since the classifier has to sequentially compare all the five fields of the packet header with the filtering rules, it can predefine an order for these fields to lookup, and may find the matched filtering rule before all the five fields are looked up.

Now, how to design this pre-defined lookup order to minimize the classification time is an important issue. There are two ways to determine the lookup order. One is to try the $5! = 120$ combinations of lookup orders and select the one that has minimal classification time, which is impractical. The other is that the classifier considers the distribution of the filtering rules for each field, selects the field where the filtering rules are distributed most evenly, and compares the packet against that field first. We will describe the strategies to determine the lookup order according to the distribution of the filtering rules in Section 3.3.

3. Ordered lookup with bypass matching

3.1. Data structures of the classification database

The date structures of our classification database are constructed by two primitive tables, named 64k-table and 256-table, as shown in Fig. 2. The detailed data structures for each field are shown in Fig. 3. The index of the tables corresponds to the value of the field. Each table entry stores

Table 1
Summary and comparison of several classification methods

	High-speed policy based packet forwarding using efficient multi-dimensional range matching [5]	Fast and scalable layer 4 switching [6]	Fast packet classification using Tuple Space Search [7]
Method	Multi-dimension matching, bit-parallelism	Dest-src tries, cross-producting	Tuple Space Search, Tuple pruning
Style	Hardware	Software	Software
Scalability	Thousands to tens of thousands of filters	Tens of thousands of filters	Not validated beyond 278 filters
Throughput	About 1 Mpps	At least 1 Mpps	N/A
Features	Using special hardware, low memory space	Using general processor, high memory space	Fast database update time

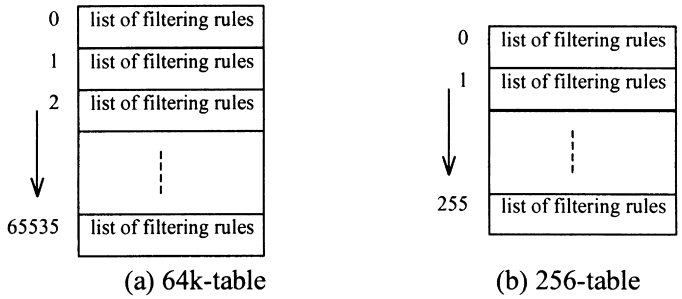


Fig. 2. The data structure of the 64k-table and the 256-table.

a list of 3-byte pointers to the filtering rules in the filter database.

The data structures for the fields of src/dest IP address require more explanations. The index value of each entry in the first 64k-table represents a 16-bit IP prefix. The index value of each entry in the second 64k-table represents a 16-bit IP suffix.

3.1.1. Insertion operations

A filtering rule is inserted into an entry of the table for a specific field, according to its value for that field. The insertion operation for src/dest IP address field needs more explanations. When a filtering rule is to be inserted into the classification database, its IP address field is split into two parts: a 16-bit IP prefix and a 16-bit IP suffix. The filtering rule is inserted into the first and the second 64k-table using the prefix part and the suffix part of the IP address, respectively. The advantage of this scheme is that we do not have to use a 32-bit flat table but needs merely two 64k-tables to index an IP address field. The disadvantage is that the lookup operation is a little more complex than that of the 32-bit flat table.

Now we give an example to show how to insert filters into our classification database. Table 2 lists five examples of filtering rules to be inserted into the classification database. Fig. 3 shows the classification database after inserting filtering rules from Table 2.

3.1.2. Lookup operations

To clarify the description of the lookup operation, we first define some terms.

Definition: ordered lookup

Ordered lookup is the process of performing a *lookup*

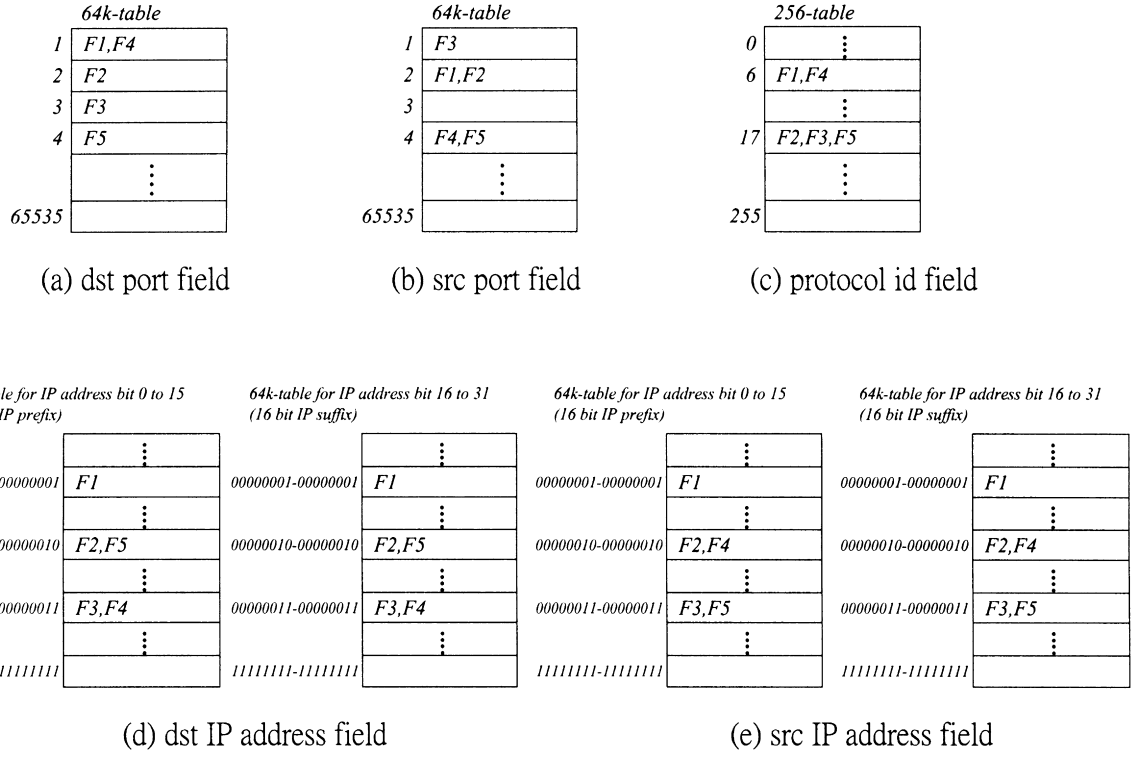


Fig. 3. The classification database after inserting filtering rules from Table 2.

Table 2
The example filtering rules for insertion operation

	Dest port	Src port	Dst IP	Src IP	Prot.
Filter 1	1	2	1.1.1.1	1.1.1.1	6
Filter 2	2	2	2.2.2.2	2.2.2.2	17
Filter 3	3	1	3.3.3.3	3.3.3.3	17
Filter 4	1	4	3.3.3.3	2.2.2.2	6
Filter 5	4	4	2.2.2.2	3.3.3.3	17

operation with the five fields of a packet, one by one, in a pre-defined order.

Definition: lookup operation

A *lookup* operation is to take the value of one field of a packet header as the index into the tabular data structures for the field.

Definition: match operation

A *match* operation is to compare all the five fields of a packet with the corresponding fields of one filtering rule.

The advantage of using tables is fast access to the field-matched filtering rules through indexing. As shown in Fig. 3, the port number value or protocol identifier value are used as indexes to the tables. When the classifier wants to get the field-matched filtering rules for the source/destination IP address, it uses the 16-bit prefix part and 16-bit suffix part of the IP address in the packet header as indexes to the two basic tables, respectively, to retrieve the corresponding filtering rules. If the sets of filtering rules found in the first 64k-table and the second 64k-table are denoted as FR1 and FR2, the classifier performs the operation (FR1 $\cap$ FR2). The result of the operation is the field-matched filtering rules collected for the IP address field.

3.2. Ordered lookup

We introduce the Ordered Lookup algorithm here. First, we define some terminologies to clearly describe the algorithm.

Definition: field-matched filtering rule

A *field-matched filtering rule* is a filter retrieved by looking up only one field of the packet header.

Definition: matched filtering rule

A *matched filtering rule* is filter that contains exactly the same values for the five fields as the incoming packet at the end of the classification process.

Table 3
Function definition of our algorithm

Function	Description
Lookup(PF,FF)	Lookup the field PF of a packet against the tabular data structures for the field FF
Match(PKT,FR)	Match a packet PKT with a set of filtering rules FR

Definition: partially-matched filtering rule

A filter is said to be a *partially-matched filtering rule* if some of its fields are the same as those in the incoming packet.

Following is the description of the Ordered Lookup algorithm.

The classifier *lookups* the first field of a packet, specified by a pre-defined order, against the tabular data structures and obtains a set of partially-matched filtering rules, which forms a *candidate* set of *matched filtering rules*. If the size of the set is 0, surely the packet does not match any filtering rules; if the size of the set is 1, which means the packet may or may not match that filtering rule, the classifier performs a *match* operation with the packet and that filtering rule to see whether the packet really matches the filtering rule; if the number of the set is more than 1, the classifier continues to *lookup* the second field and obtains another set of filtering rules. If the number of this set is more than 1, the classifier intersects this set with the candidate set and results in a new candidate set of filtering rules. If the new candidate set contains only one filtering rule, a direct *match* is performed; if there are more than one filtering rule, subsequent lookups and intersections are performed.

Tables 3 and 4 describe the required functions and variables in our pseudocode. Fig. 4 shows the pseudocode of this algorithm.

Fig. 5 gives some example operations of the Ordered Lookup algorithm. There are five filters in the classification database. When packet A comes, the classifier uses two lookups and finds that it matches filter 1. Packet B matches filter 3 with only one lookup. The classifier uses two lookups and finds that packet C does not match any filters.

3.2.1. Decision strategies for lookup order

It may happen that different lookup orders will result in different classification speeds. The best lookup order is the one that uses the least lookups on the average to find the matched filtering rule, if any. To minimize the number of lookups, it is straightforward that the classifier should first lookup the tabular data structures for the field in which the average number of field-matched filtering rules per table entry is the least among all the five fields. Because the

Table 4
Variable description of our algorithm

Variable	Description
Packet	The incoming packet
LO[0–4]	The lookup order of the five fields
CFR[]	The candidate set of filters rules for a packet
FR[]	One-field lookup resulting set of filters for a packet
Destination_port	The tabular data structures for the dest port number
Destination_IP	The tabular data structures for the dest IP address
Source_port	The tabular data structures for the src port number
Source_IP	The tabular data structures for the src IP address
Protocol_id	The tabular data structures for the protocol

```
Algorithm: Ordered_Lookup
Input: packet, LO[0~4]     /* lookup order */
Output: CFR[ ]         /* candidate set of filtering rules */

CFR[ ] = NULL
for I = 0 to 4
  switch LO[I]  /* lookup the 5 fields by the given order */
      case Destination_Port:
        FR[ ] = lookup(packet.destination_port, Destination_port)
        goto Match
      case Destination_IP:
        FR[ ] = lookup(packet.dstination_IP,  Destination_IP)
        goto Match
      case Source_Port:
        FR[ ] = lookup(packet.source_port, Source_port)
        goto Match
      case Source_IP:
        FR[ ] = lookup(packet.source_IP,  Source_IP)
        goto Match
      case Protocol_Identifier:
        FR[ ] = lookup (packet.protocol_id, Protocol_id)
        goto Match
      Match:
        if sizeof FR[ ] = 0, return NULL
        if sizeof FR[ ] = 1 and match(packet, FR[ ]) = TRUE, return FR[ ]
        else CFR[ ] = CFR[ ] ∩ FR[ ]
        if I=4, return CFR[ ]    /* last lookup order */
        if sizeof CFR[ ] = 1 and match(packet, CFR[ ]) = TRUE, return CFR[ ]
  end switch
end for
```

Fig. 4. Pseudocode of ordered lookup algorithm.

classifier might get the minimal number of field-matched filtering rules on the average after each lookup, the lookup sequence is more likely to terminate midway.

There are other ways to determine a good lookup order. We define the average number of field-matched filtering rules per entry as *avg*, and the standard deviation of the number of field-matched filtering rules per entry as *sdv*. We provide the following three strategies for the classifier to determine the lookup order. The lookup order is according to the sorted results, from minimum to maximum, of the five fields.

MAF: minimum average-length first

The classifier first lookups the tabular data structures for the field that has the minimal *avg*. Thus the classifier is likely to get the least field-matched filtering rules on the average after each lookup. However, in the worst case the classifier may index to the entry that has the maximum number of filtering rules among all entries. This maximum number could be large if *sdv* of the field is large.

MSF: minimum standard deviation first

The classifier first lookups the tabular data structures for the field that has the minimal *sdv*. Thus the classifier is likely to get a similar number of field-matched filtering rules after each lookup.

MASF: minimum average and standard deviation field first

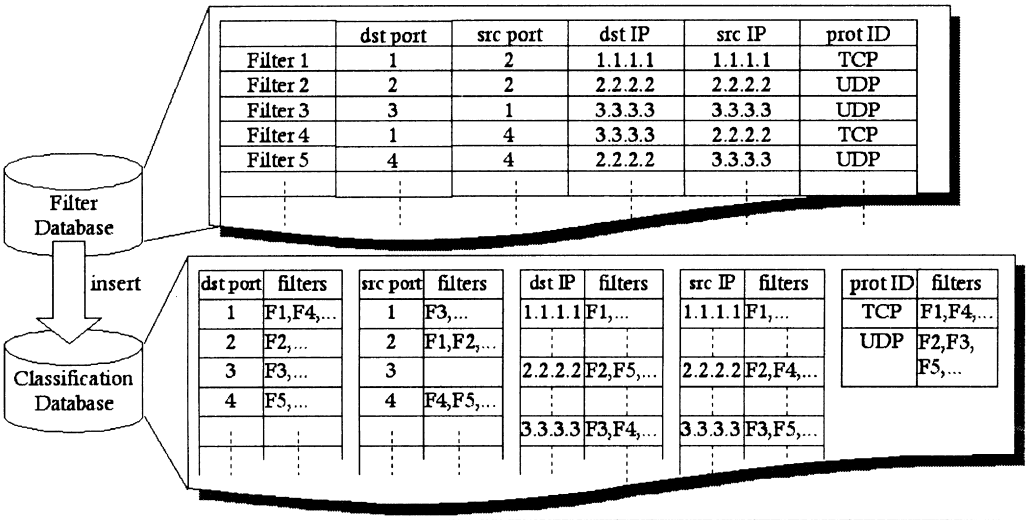
The classifier first lookups the field that has the minimal *avg·sdv*. That means the field has the minimal product of the average and standard deviation of the number of filtering rules per entry. Thus the classifier may, not only on the average case but also on the worst case, get the least field-matched filtering rules after each lookup.

We compare the throughput of different lookup orders determined by these three strategies in Section 5.

3.3. Bypass matching

As mentioned in Section 3.2, the classifier terminates the lookup process by a direct match when the number of field-matched or partially-matched filtering rules is one. In fact, the lookup process might be terminated more quickly. Assume that the number of field-matched or partially matched filtering rules is k; then the classifier matches the packet directly with the k field-matched or partially-matched filtering rules if the cost of k matches is less than that of the remaining lookups. We call this operation *bypass matching*, and define the maximum value of k that satisfies the above criteria for direct matching as K. Fig. 6 is the modified pseudocode for the OLBM algorithm.

The threshold K for bypass matching is a machine dependent threshold. We now describe how to determine the bypass matching threshold. There are two major kinds of CPU instructions in our algorithm. One is *memory reference*, and the other is *compare*, with costs Cm and Cc. If there are k filters left after lookup of some fields, whether to directly match k filters or to continue to lookup



	dst port	src port	dst IP	src IP	prot ID
Filter 1	1	2	1.1.1.1	1.1.1.1	TCP
Filter 2	2	2	2.2.2.2	2.2.2.2	UDP
Filter 3	3	1	3.3.3.3	3.3.3.3	UDP
Filter 4	1	4	3.3.3.3	2.2.2.2	TCP
Filter 5	4	4	2.2.2.2	3.3.3.3	UDP

dst port	filters
1	F1,F4,...
2	F2,...
3	F3,...
4	F5,...

src port	filters
1	F3,...
2	F1,F2,...
3	
4	F4,F5,...

dst IP	filters
1.1.1.1	F1,...
2.2.2.2	F2,F5,...
3.3.3.3	F3,F4,...

src IP	filters
1.1.1.1	F1,...
2.2.2.2	F2,F4,...
3.3.3.3	F3,F5,...

prot ID	filters
TCP	F1,F4,...
UDP	F2,F3, F5,...

Example 1	dst port	src port	dst IP	src IP	protocol ID
Incoming packet A	1	2	1.1.1.1	1.1.1.1	TCP
After lookup 1st field	(F1,F4)				
After lookup 2nd field	(F1,F4)	(F1,F2)			
Intersection	(F1,F4)∩(F1,F2) = (F1)				
match	Match(packet A, F1)=TRUE, so packet A matches F1				

Example 2	dst port	src port	dst IP	src IP	protocol ID
Incoming packet B:	3	1	3.3.3.3	3.3.3.3	UDP
After lookup 1st field:	(F3)				
Match:	Match(packet B, F3)=TRUE, so packet B matches F3				

Example 3	dst port	src port	dst IP	src IP	protocol ID
Incoming packet C:	1	3	4.4.4.4	4.4.4.4	TCP
After lookup 1st field:	(F1,F4)				
After lookup 2nd field	(F1,F4)	NULL			
Intersection	(F1,F4)∩NULL = NULL, so packet C doesn't match any filtering rule				

Fig. 5. Examples for the ordered lookup algorithm.

remaining fields is up to the following inequality:

$$k(\mathrm{Cm} + 5\mathrm{Cc}) < 2\mathrm{Cm} + k\frac{N}{E}\mathrm{Cc} + (\mathrm{Cm} + 5\mathrm{Cc})$$

where E is the number of entries in the tables for a field and N is the number of filters.

The left part of the inequality is the cost of k direct matchings, each of which contains one memory reference to retrieve the filter from the filter database and five comparisons to compare the five fields. The right part of the inequality is the least cost to lookup the remaining fields, which happens in the following situation: the next field lookup first uses two memory references to take the next field of the packet to index to the corresponding table, and retrieves (N/E) field-matched filtering rules, in the average case. Then the (N/E) filters intersect with the k filters to form

```
Match:
  if sizeof FR[ ] = 0 return NULL
  if sizeof FR[ ] <= K
      for each J in FR[ ]
            if match(packet, J) = FALSE
                  remove (J, FR[ ])
      return FR[ ]
  else CFR[ ] = CFR[ ] ∩ FR[ ]
  if I=4, return CFR[ ]    /* last lookup order */
  if sizeof CFR[ ] <= K
      for each J in CFR[ ]
            if match(packet, J) = FALSE
                  remove (J, CFR[ ])
      return CFR[ ]
```

Fig. 6. Modified pseudocode for OLBM.

Table 5
The time and space comparison of three classification methods

Description	Time complexity	Space complexity
Ordered Lookup with Bypass Matching	O($N{\cdot}F$)	O($N{\cdot}F$)
High-speed policy based packet forwarding using efficient multi-dimensional range matching [5]	O($N{\cdot}F$)	O($N^2{\cdot}F^2$)
Fast and scalable layer 4 switching [6]	O(log W + (W/k))	O($N{\cdot}W$)
Fast packet classification using Tuple Space Search [7]	O(W)	O($N{\cdot}W$)

a new candidate set of matched filtering rules, which costs $k(N/E)$Cc. The least cost situation happens where the resulting candidate set contains only one filter, and its remaining cost is simply a match operation of that filter, which costs (Cm + 5Cc).

Thus,

$$k < \frac{3\mathrm{Cm} + 5\mathrm{Cc}}{\mathrm{Cm} + \left(5 - \frac{N}{E}\right)\mathrm{Cc}}$$

So the maximum number of k, denoted by K, is equals to

$$\left\lfloor \frac{3\mathrm{Cm} + 5\mathrm{Cc}}{\mathrm{Cm} + \left(5 - \frac{N}{E}\right)\mathrm{Cc}} \right\rfloor.$$

In brief, once you have decided to run this algorithm on some machine, given the costs of memory references and compare instructions, with the knowledge of the current number of filtering rules, N, and the current number of entries in the table, the machine dependent threshold K can be determined.

Let the bypass matching threshold be two for the example in Fig. 2. When packet A arrives, the classifier uses only one lookup and two matches, and finds that the packet belongs to filter 1. As for packet C, the classifier uses one lookup and two matches and finds that packet C does not match any filters.

4. Complexity analysis: time and space

Our Ordered Lookup with Bypass Matching algorithm is concerned with five fields. It has at most five lookups for the five fields in the classification process. Because each field is basically the same, we show the time and space complexity of this algorithm by analyzing one field.

Let us look at Fig. 3(a) and (b). The lookup operation in the figure is where the classifier takes the port number as an index and retrieves the field-matched filtering rules. This operation takes O(N) in the worst case where N is the total number of filtering rules. The worst case happens when all of the filtering rules in this field are stored in only one table entry. After retrieving the field-matched filtering rules, the classifier either lookups into the next field and then intersects the resulting field-matched filtering rules with those from the previous lookup, if any, or matches the packet with the currently candidate set of matched filtering rules. Both of them take O(N) time in the worst case.

As we have mentioned that there are at most five lookups and four matches in our method, the time complexities of lookups and matches for F fields are O($N{\cdot}F$) and O($N{\cdot}(F-1)$). And the time complexity of our algorithm is O($N{\cdot}F$).

The minimal space requirement happens when all the filtering rules stored in the classification database specify exact values because each filtering rule is stored in exactly one table entry in each field. Thus the classification database requires $N{\cdot}F{\cdot}3$ bytes, where 3 is the size of a pointer to one filtering rule. But the maximum requirement happens when all the filtering rules are of a range type or prefix type that needs to be expanded. Then the space requirement becomes $N{\cdot}F{\cdot}M$, where M is the maximum number of expanded entries over F fields. Thus the space complexity for our classification database is O($N{\cdot}F$). Note that the situation where our classifier requires maximum space rarely happens. The space requirement for our classification method in the Integrated Service networks, with per-flow filtering rules, always approximates the minimum.

Table 5 shows the time and space complexity of three classification methods. The time complexity of our method

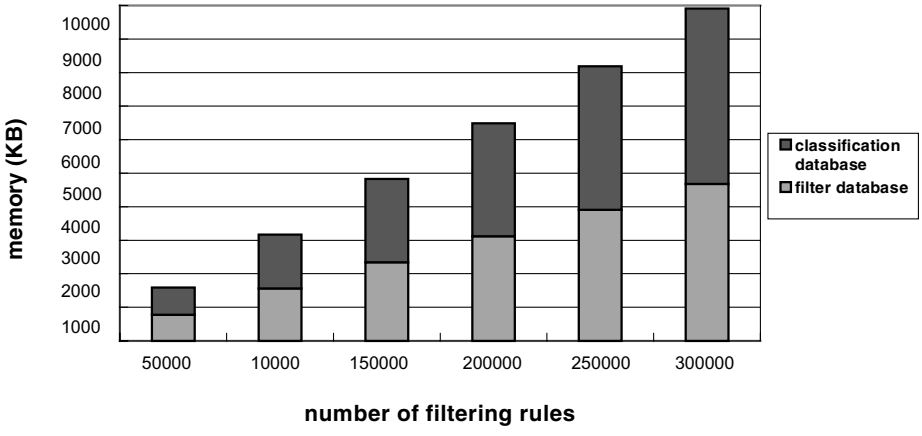


Fig. 7. Memory usage of the ordered lookup algorithm.

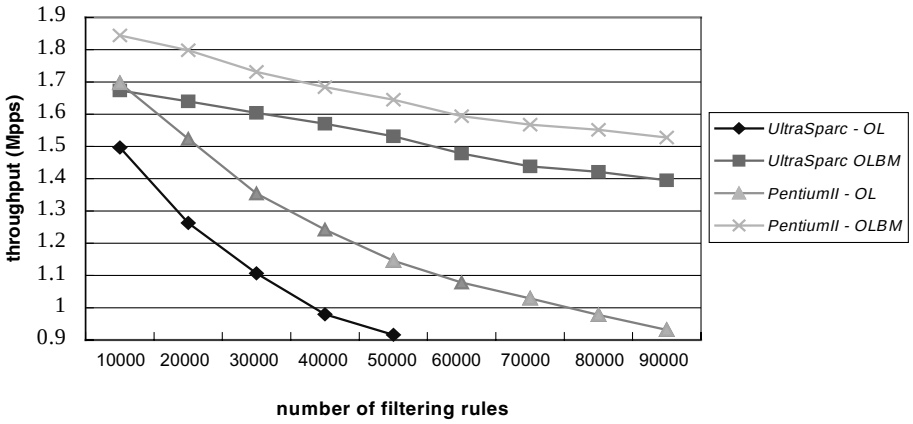


Fig. 8. The effect of ordered lookup and bypass matching on two platforms.

is the same as the one in Ref. [5]. But the space complexity of our method is much lower than the one of Ref. [5]. The method of Ref. [6] needs very large memory space and cannot scale to 300k filtering rules (W denotes the maximum bit length of any destination or source prefix, k denotes the number of fields to be checked by the classifier).

5. Performance study

We have implemented and experimented our algorithm on two platforms. One is the Intel Pentium-II 350 Mhz CPU platform and the other is the Sun UltraSparc 300 Mhz CPU platform. The hit ratio for arriving packets is 80%, i.e. 80% of arriving packets will hit one filtering rule and the remaining 20% will not. The default strategy for deciding lookup order is MAF. In this section we show the numerical results on these two platforms plus some implementation issues.

5.1. Memory usage

Fig. 7 shows the memory usage of our algorithm. We randomly generate these filtering rules and add them into the classification database. The portion of classification database in Fig. 7 is the total size of memory usage for the tabular data structures for all the five fields. And the portion of the filter database is the size of memory used to store the filtering rules. Our classification algorithm uses quite a reasonable amount of memory, e.g. 10 MB for 300k filtering rules. The algorithm by Srinivasan et al. [6] uses 7.489 MB memory for only 20k filtering rules.

5.2. Throughput

Fig. 8 shows the throughput of the ordered lookup and the improvement of bypass matching. In this experiment the filtering rules are randomly generated. Note that without bypass matching the throughput drops more rapidly because the number of lookups for each packet increases when the number of filtering rules increases. We can see that from Fig. 9.

Considering the memory references and compare costs on the two selected platforms, we set the bypass matching threshold to two.

5.3. Sensitivity to locality

In routers, both *incoming packets* and *filtering rules* might have some degree of address locality and could affect the performance of the classifier. We use three simple address locality models to simulate the address patterns observed at a router.

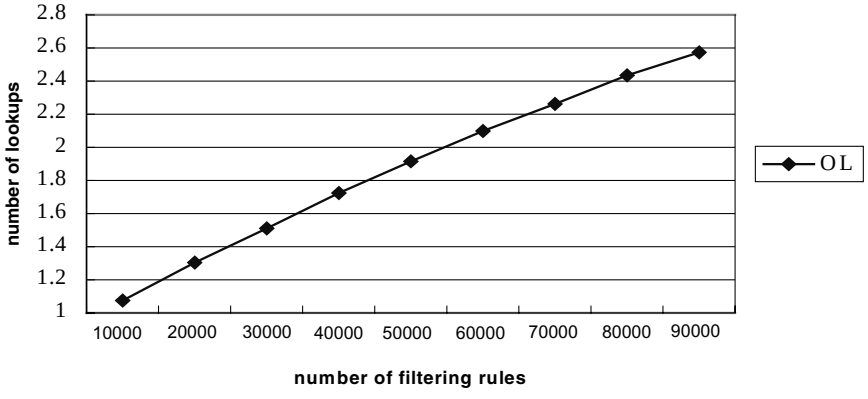


Fig. 9. Number of lookups used by the ordered lookup algorithm.

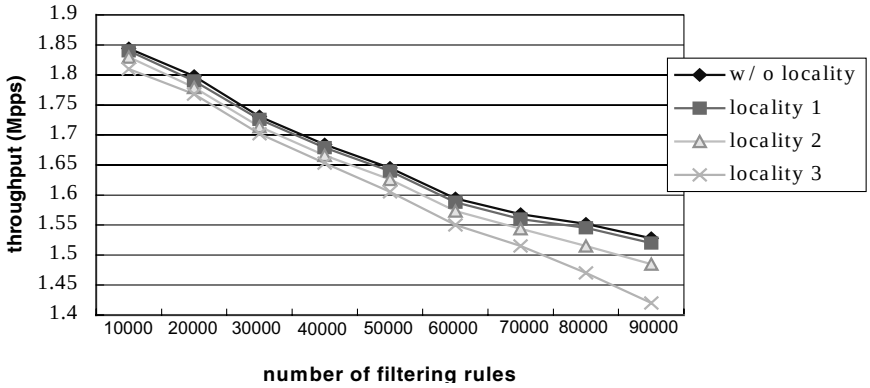


Fig. 10. The effect of address locality (on Pentium-II platform).

Considering each field except the protocol identifier field, for the first locality model, we let 80% of the filtering rules generated concentrate in 30% of the address space of that field, and the remaining 20% of the filtering rules are randomly generated. The second locality model is that we let 80% of the filtering rules concentrate in 20% of the address space of that field, and the remaining 20% are randomly generated. The third locality model is that we let 80% of the filtering rules concentrate in 10% of the address space of that field, and the remaining 20% are randomly generated. From Fig. 10 we can see that higher address locality leads to lower throughput, especially when the number of filtering rules is large. It is because the average number of field-matched filtering rules after each lookup increases, as both the address locality and the number of filtering rules increase. But overall, the throughput sensitivity to locality is not very high.

5.4. Decision strategy for lookup order

We use the first locality model to generate the filtering rules for this experiment. In Section 3.3 we provided three decision strategies for lookup order. In Fig. 11 it is seen that MSF and MASF result in almost the same throughput. The lookup orders decided by these two strategies are almost the same in the repeated runs. The standard deviation appears to be the more dominant factor than the average. But the throughput of MAF is a little lower, because of the worst case we described in Section 3.3. MSF and MASF turn out to be better strategies for our algorithm.

5.5. Scalability

Fig. 12 shows that our Ordered Lookup with Bypass Matching algorithm is scalable. Even with 300k filtering rules in the classification database, the throughput is still above 1.1 Mpps, compared to 1.1 Mpps with 20k filtering rules in Ref. [6]. 300k is already larger than the design objective, 240k [8], set in Section 2. The bypass matching threshold is set to 8 for this experiment. When the number of filtering rules increases, the number of filtering rules per table entry becomes larger, which increases the cost of intersection operations. The intersection operation dominates the performance of our algorithm.

5.6. Extensibility

Although our algorithm is mainly for per-flow classification, it can be extended to lookup more fields or even the payload of a packet. Once you decide to extend it to support more fields, you simply analyze the extra memory reference

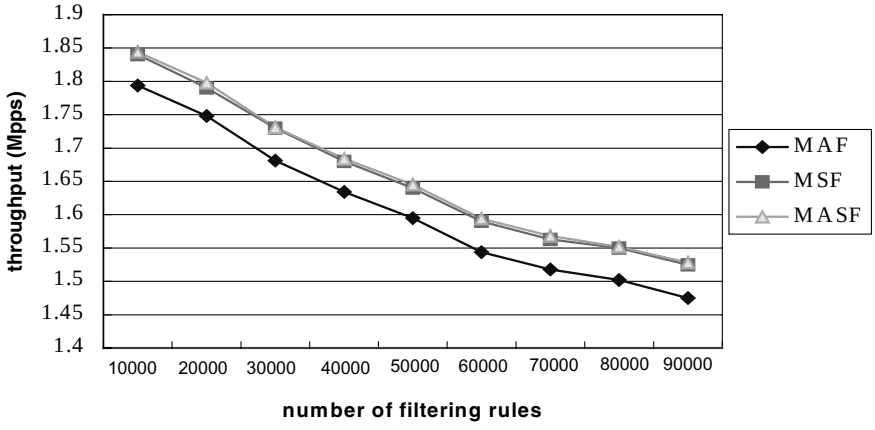


Fig. 11. The effect of different lookup order decision strategies on Pentium-II platform.

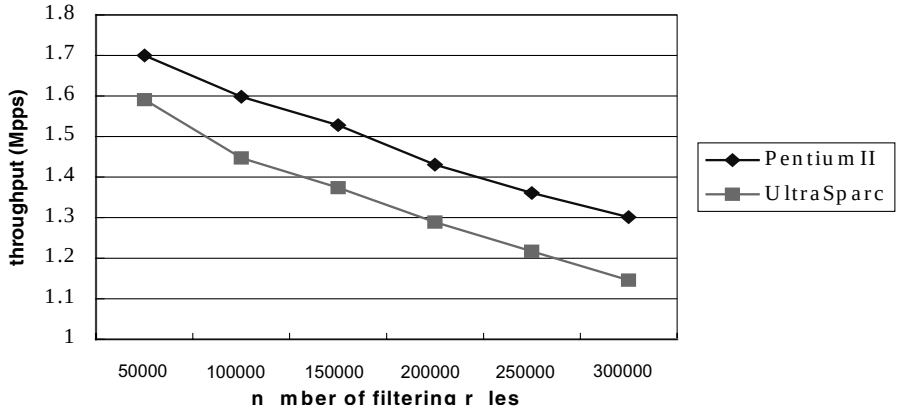


Fig. 12. At least one million packets per second for 300k filtering rules.

and compare instructions to lookup that field, and then recompute the machine dependent threshold K as the criterion for bypass matching. In this way, the algorithm is capable of being extended to lookup various filtering rules for VPN or firewall layer 4 routers, while preserving high performance features such as scalability and throughput.

5.7. Implementation issues

There are some important issues that should be noted during implementation. When this algorithm is implemented in a Linux router, since the kernel memory cannot be paged out, the data structures should be as compact as possible. Sufficient physical memory should be available to support the scale of filtering rules. In the control plane the filter updating process can be written as a daemon and can update the filters in kernel space through a netlink socket (like the routing socket in BSD).

When this algorithm is implemented in a router device, some network processor could be adapted to speed up some operations such as intersection. The throughput and scale could be largely enhanced by doing intersection in an O(1) fashion.

6. Conclusions

In this work we presented a new multi-field classification algorithm. Our Ordered Lookup with Bypass Matching algorithm can dynamically determine the lookup order according to the length distribution of filtering rules in the table entries for each field. Following the pre-defined lookup order the classifier lookups tables for a packet and may find the matching filtering rule without looking up tables for all the five fields. It also uses bypass matching to terminate the lookup process when direct match with the current matching filtering rules is more cost effective. This algorithm is traffic-aware and adaptive. It can be extended to classify packets based on more fields.

This algorithm scales well to support 300k filtering rules using 10 MB memory at a rate over one million packets per second, compared to 1.1 Mpps with 20k filtering rules in Ref. [6]. When there are 20k filtering rules in our classification database, our algorithm is 50% faster than the algorithm proposed by Srinivasan et al.[6], which needs a very large memory space and cannot scale to 300k filtering rules. A recently proposed algorithm RFC (Recursive Flow Classification) [9] can achieve up to 31.25 million packets per second, but it cannot scale well over 6k filters. Even with its optimization scheme, it can only scale to 15k filters.

There is some future work to be done. First, we plan to embed and integrate our algorithm into a layer 4 router to see its performance, and how it interacts with other modules in the router. Second, we will try to extend our algorithm to classify packets based on more fields.

References

[1] J. Wroclawski, The use of RSVP with IETF integrated services, RFC 2210, September 1997.
[2] J. Wroclawski, Specification of the controlled-load network element service, RFC 2211, September 1997.
[3] S. Shenker, C. Partridge, R. Guerin, Specification of guaranteed quality of service, RFC 2212, September 1997.
[4] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, A. Malis, A framework for IP based virtual private networks, Internet Draft, draft-gleeson-vpn-framework-01.txt, February 1999.
[5] T.V. Lakshman, D. Stiliadis, High-speed policy-based packet forwarding using efficient multi-dimensional range matching, in: Proc. ACM Sigcomm'98, September 1998.
[6] V. Srinivasan, G. Varghese, S. Suri, M. Waldvogel, Fast and scalable layer four switching, in: Proc. ACM Sigcomm'98, September 1998.
[7] V. Srinivasan, S. Suri, G. Varghese, Packet classification using tuple space search, in: Proc. ACM Sigcomm'99, September 1999.
[8] K. Thompson, G.J. Miller, R. Wilder, Wide-area Internet traffic patterns and characteristics, IEEE Network 11 (6) (1997) 10–23.
[9] P. Gupta, N. McKcown, Packet classification on multiple fields, in: Proc. ACM Sigcomm'99, September 1999.